The Greedy method

General method:

It is the process of making local optimal choice at every stage to get the global optimal solution

It is used for solving optimization problem. Given *n* inputs choose a subset that satisfies some constraints.

- **A subset that satisfies the constraints is called a feasible solution.**
- **A feasible solution that maximises or minimizes a given (objective) function is said to be optimal.**

Note: Often it is easy to find a feasible solution but difficult to find the optimal solution. The greedy method suggests that one can devise an algorithm that works in stage. At each stage a decision is made whether a particular input is in the optimal solution. This is called **subset paradigm**.

1 Example: Your Train breaks down in a desert and you decide to walk to nearest town. You have a rucksack but which objects should you take with you?

Feasible: Any set of objects is a feasible solution provided that they are not too heavy, fit in the rucksack and will help you survive (these are constraints).

An optimal solution is the one that maximizes or minimises something – One that minimises the weight carried

– One that fills the rucksack completely (maximise)

– One that ensures the most water is taken etc.

2 Example: You are writing a computer game and need to store your .wav, .jpg, and .mpg files on a set of CD-roms. The constraint is the length of files and the **size of the CD (capacity)** A feasible solution is any combination that fits.

An optimal one is the order that minimises the access time for each file.

– In this case you will have the minimum delay for the player.

– Hence you might sell more games.

– Have smoother game action.

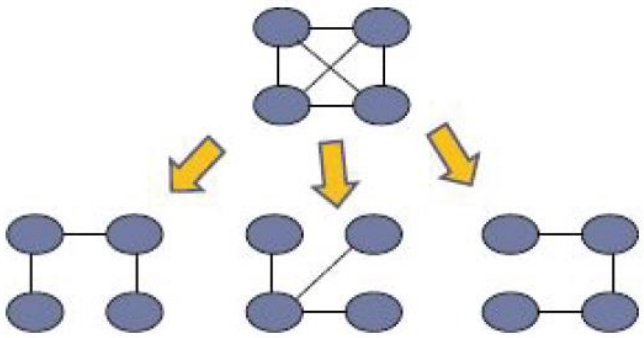
Control Abstraction for Greedy algorithm

```
Algorithm Greedy(A : set; n : integer)
{
 MakeEmpty(solution);
        for(i = 2; i <= n; i++)
   {
      x = Select(A);
          if Feasible(solution; x) then
          solution = Union(solution; {x})
    }
  return solution
}
```

The function Greedy describes the essential way that a greedy algorithm will look, once a particular problem is chosen and the functions *Select*, *Feasible* and *Union* are properly implemented. The function *Select* selects an input from *A*
whose value is assign to *x*. *Feasible* is a Boolean valued function that determines if *x* can be included into the solution vector. The function *Union* combines *x* with the solution, and update the objective function.

❖ ***Spanning tree:*** Let $G = (V;E)$ be an undirected connected graph. A subgraph $T = (V;G)$ of G is a spanning tree of *G* if *T* is a tree with minimum possible number of edges.

e.g. three spanning trees of G.



– Both *T* and *G* have same set of vertices *V* .

– *T* is connected but has no cycles.

– has $(n-1)$ edges.(min edges)

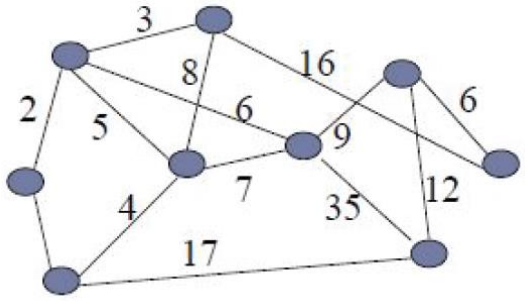
In practical applications, the edges have weights assigned to them, which may represent the cost of constructions, the length of link, etc.

Applications:

- CN Network Routing Protocol
- Cluster Analysis
- Civil Network Planning

❖ ***Minimum Spanning Tree (MST):***

Given a connected weighted graph *G*, we wish to find a set of edges with minimum (sum) weights such that all vertices are connected.

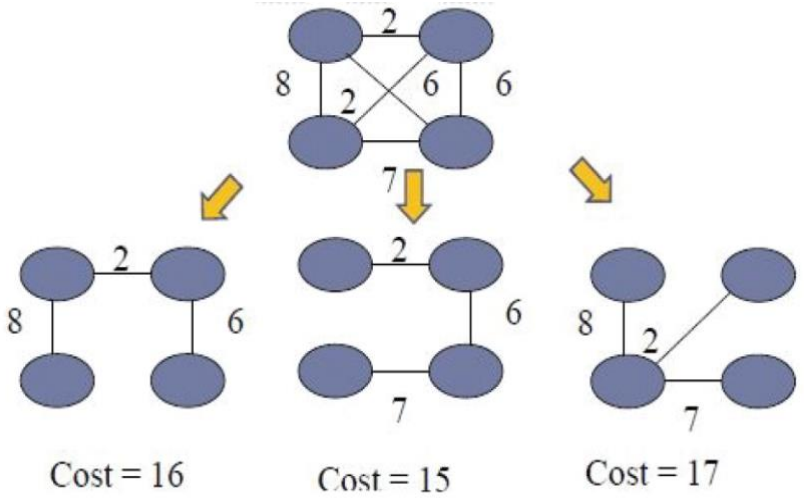


Feasible solution – The spanning trees $T = (V;E)$ represent feasible choices

Optimal solution – The spanning tree *Top t* = $(V;E)$ with the lowest total cost of edges is the one we want.

Since the identification of a minimum cost spanning tree involves the selection of a subset of edges, this problem fits the subset paradigm.

MST example



- **Solution Space:** is an set of all possible points (set of values of the choice) of an OPTIMIZATION problem that satisfy the problem constraints, potentially including inequalities, equalities and integer constraints . it is a process of backtracking.
- **Principal of Optimality** : "An optimal policy has the property that whatever the initial state and the initial decisions it must constitute an optimal policy with regards to the state resulting from the first decision."

Introduction to Prim's algorithm:

- This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.
- *The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices.*
- *The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weightedge from these edges.*
- *After picking the edge, it moves the other endpoint of the edge to the set containing MST.*
- at every step of Prim's algorithm, find a cut, pick the minimumweight edge from the cut, and include this vertex in MST Set (the set that contains already included vertices).

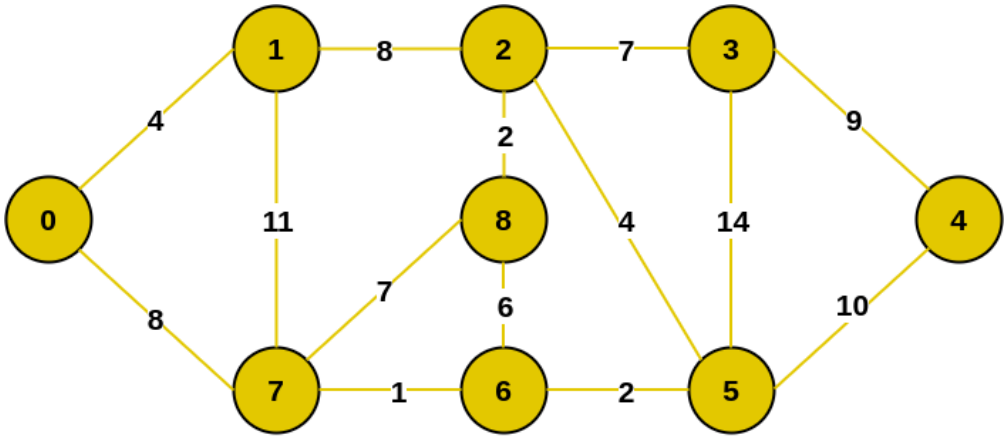
1. How does Prim's Algorithm Work?

The working of Prim's algorithm can be described by using the following steps:

Step 1: *Determine an arbitrary vertex as the starting vertex of the MST.*
Step 2: *Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).*
Step 3: *Find edges connecting any tree vertex with the fringe vertices.*
Step 4: *Find the minimum among these edges.*
Step 5: *Add the chosen edge to the MST if it does not form any cycle.*
Step 6: *Return the MST and exit*
Note: For determining a cycle, we can divide the vertices into two sets [one set contains the vertices included in MST and the other contains the fringe vertices.]

Illustration of Prim's Algorithm:

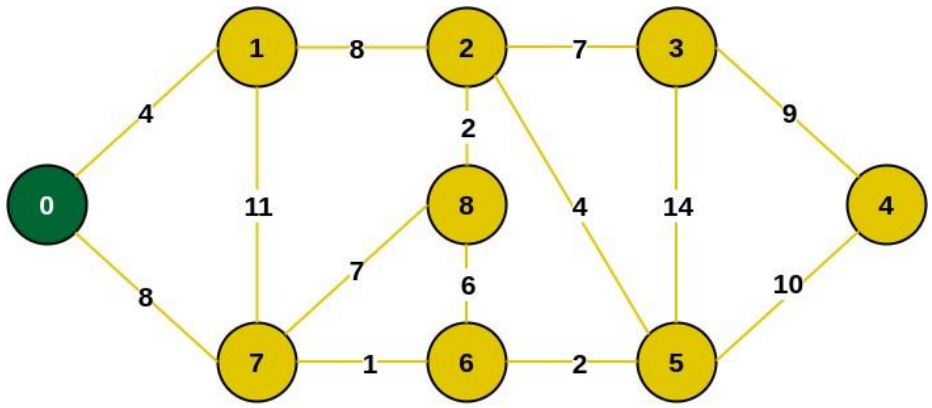
Consider the following graph as an example for which we need to find theMinimum Spanning Tree (MST).



Example of a Graph

Example of a graph

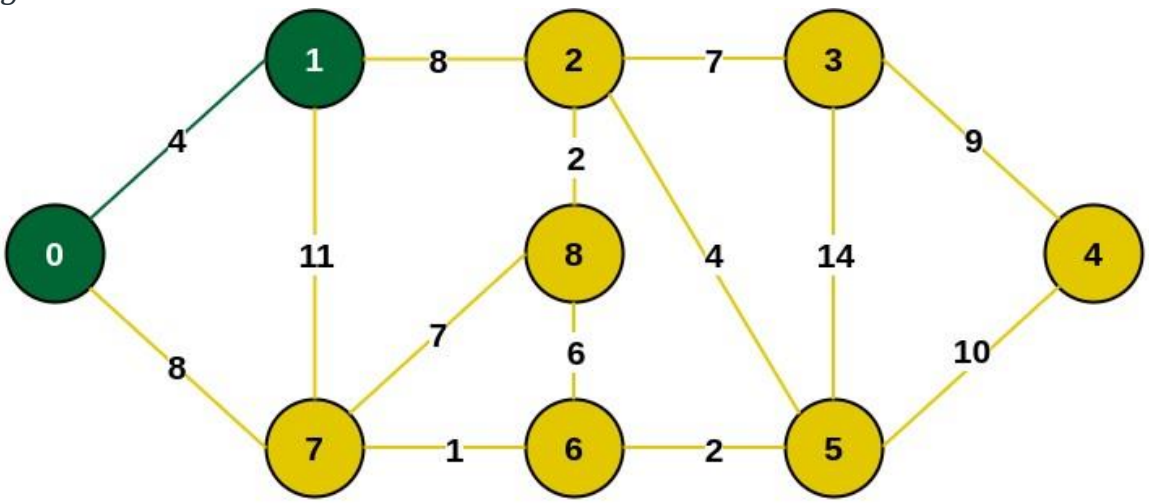
Step 1: *Firstly, we select an arbitrary vertex that acts as the starting vertex ofthe Minimum Spanning Tree. Here we have selected vertex* ***0*** *as the starting vertex.*



Select an arbitrary starting vertex. Here we have selected 0

0 is selected as starting vertex

Step 2: *All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.*

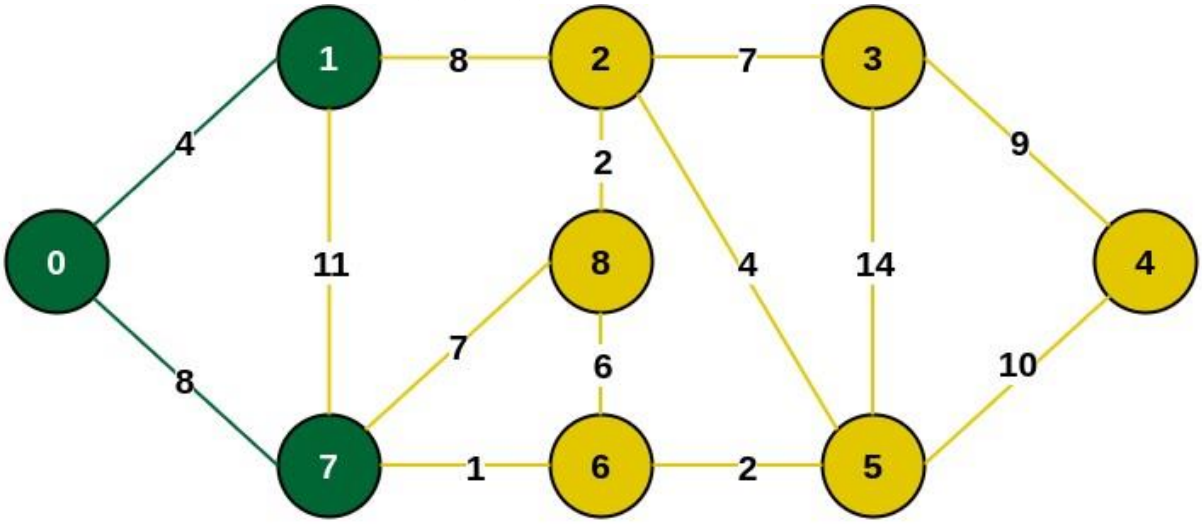


Minimum weighted edge from MST to other vertices is 0-1 with weight 4

1 is added to the MST

Step 3: *The edges connecting the incomplete MST to other vertices are {0, 7},*

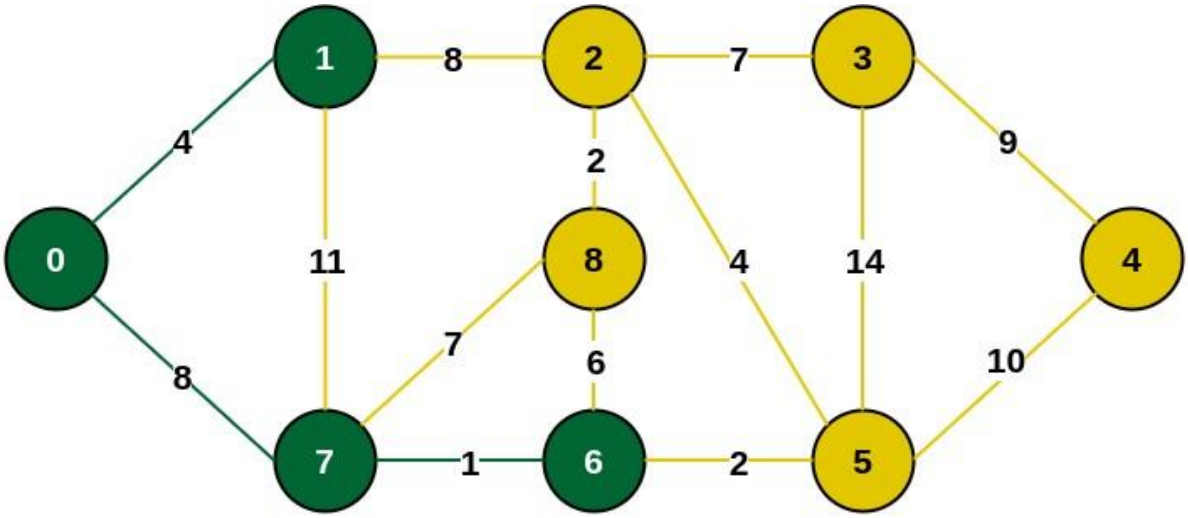
{1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].



Minimum weighted edge from MST to other vertices is 0-7 with weight 8

7 is added in the MST

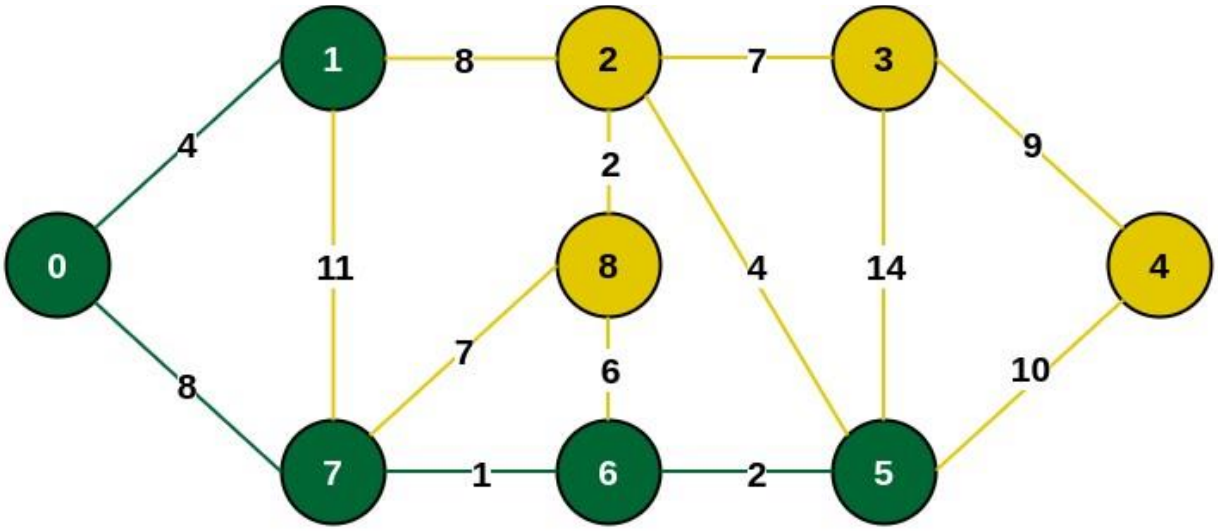
Step 4: *The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as ithas the least weight (i.e., 1).*



Minimum weighted edge from MST to other vertices is 7-6 with weight 1

6 is added in the MST

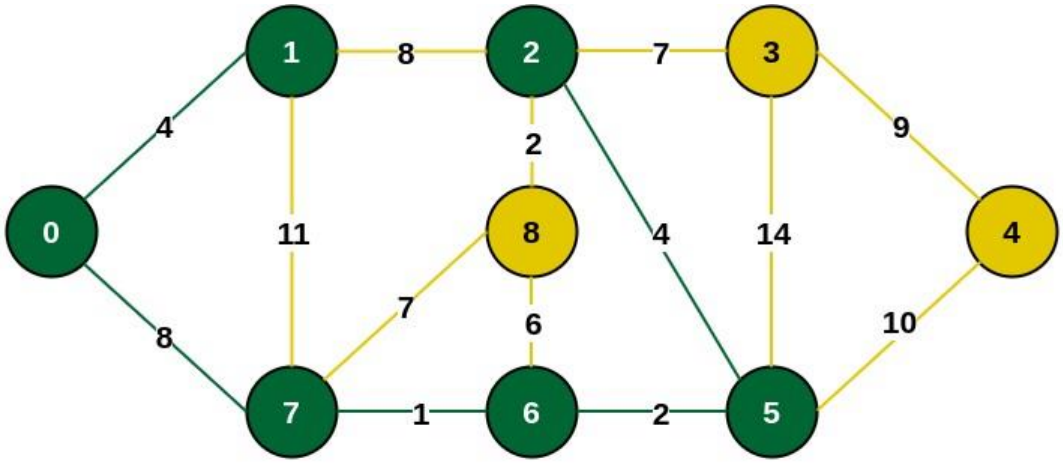
Step 5: *The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e.,2) among them.*



Minimum weighted edge from MST to other vertices is 6-5 with weight 2

Include vertex 5 in the MST

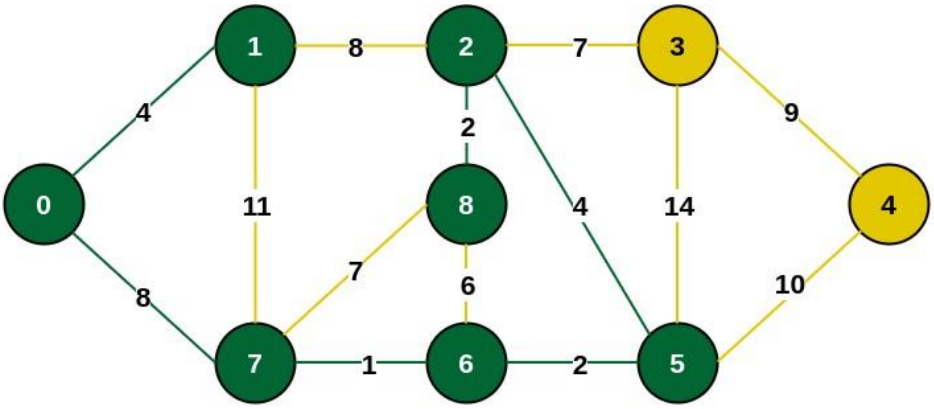
Step 6: *Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.*



Minimum weighted edge from MST to other vertices is 5-2 with weight 4

Include vertex 2 in the MST

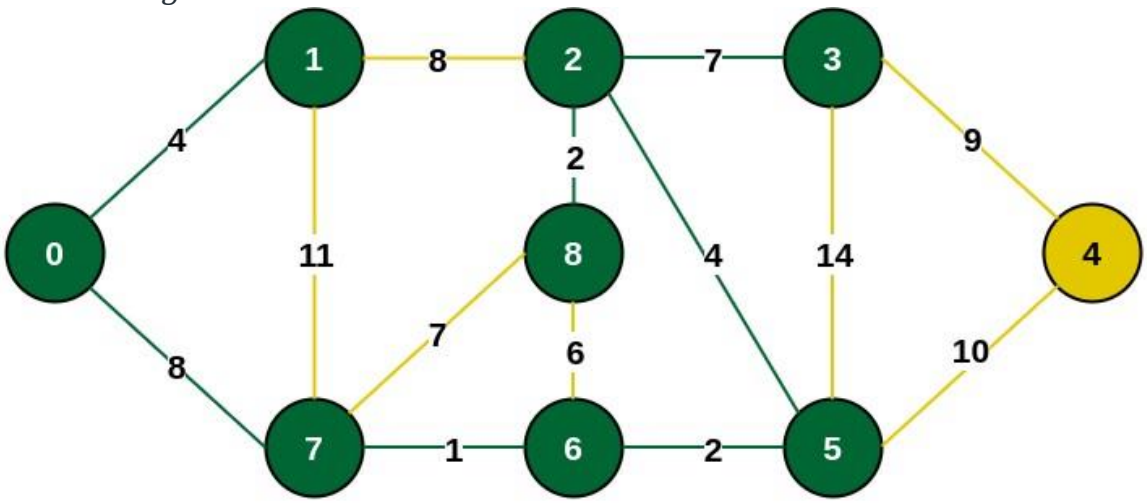
Step 7: *The connecting edges between the incomplete MST and the other edges are {2, 8}, {2, 3}, {5, 3} and {5, 4}. The edge with minimum weight is edge {2, 8} which has weight 2. So include this edge and the vertex 8 in the MST.*



Minimum weighted edge from MST to other vertices is 2-8 with weight 2

Add vertex 8 in the MST

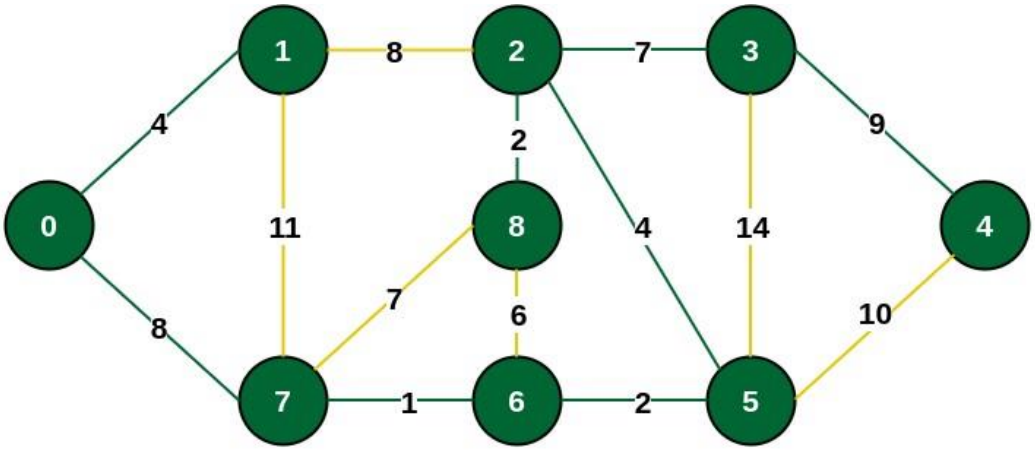
Step 8: *See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.*



Minimum weighted edge from MST to other vertices is 2-3 with weight 7

Include vertex 3 in MST

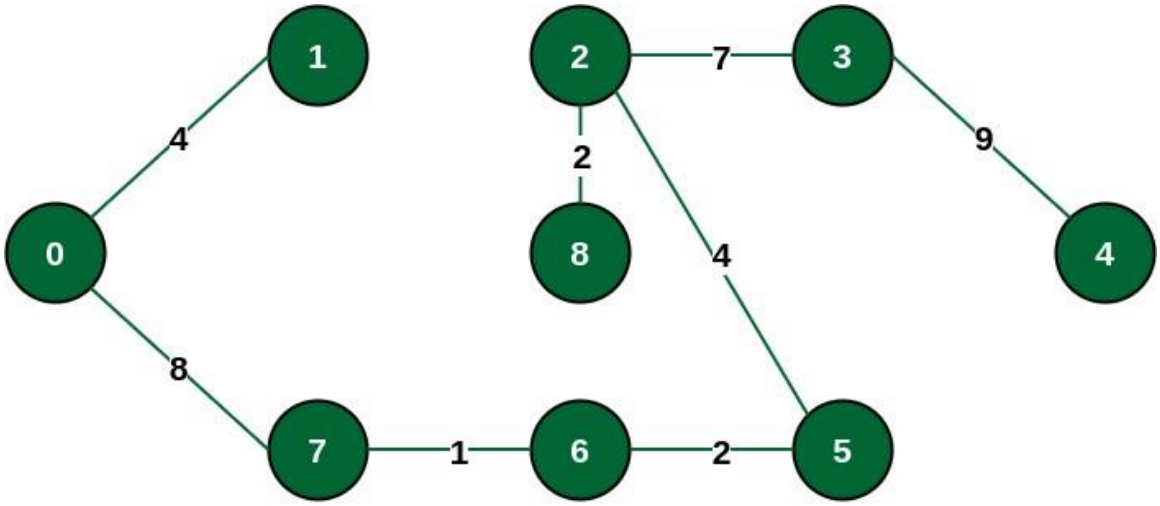
Step 9: *Only the vertex 4 remains to be included. The minimum weighted edgefrom the incomplete MST to 4 is {3, 4}.*



Minimum weighted edge from MST to other vertices is 3-4 with weight 9

Include vertex 4 in the MST

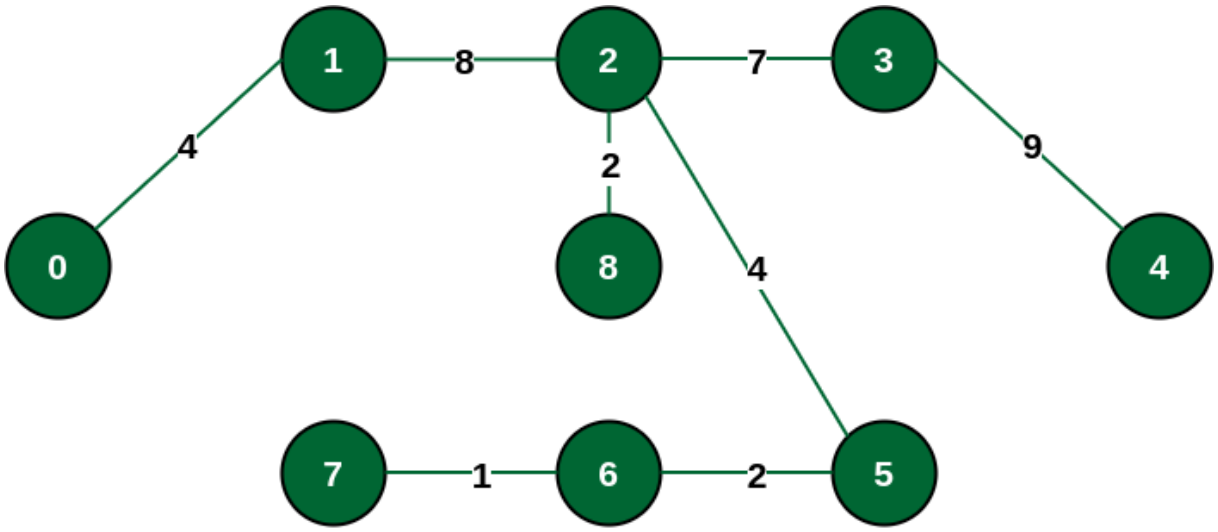
The final structure of the MST is as follows and the weight of the edges of theMST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = ***37****.*



The final structure of MST

The structure of the MST formed using the above method

Note: *If we had selected the edge {1, 2} in the third step then the MST wouldlook like the following.*



Alternative MST structure

Structure of the alternate MST if we had selected edge {1, 2} in the MST

How to implement Prim's Algorithm?

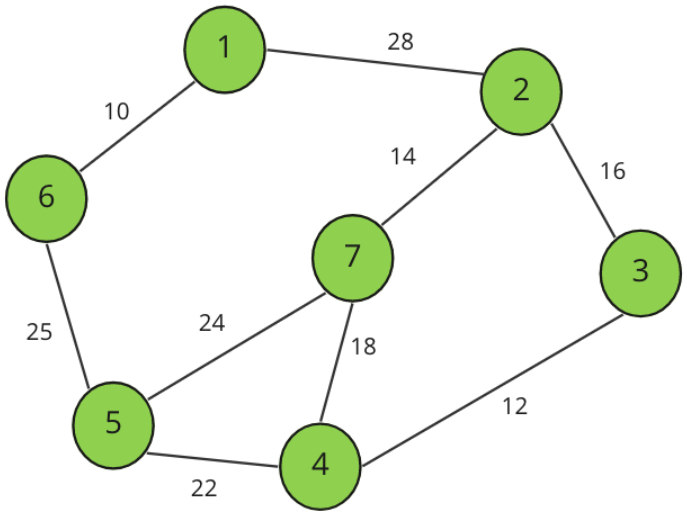
Follow the given steps to utilize the **Prim's Algorithm** mentioned above for finding MST of a graph:

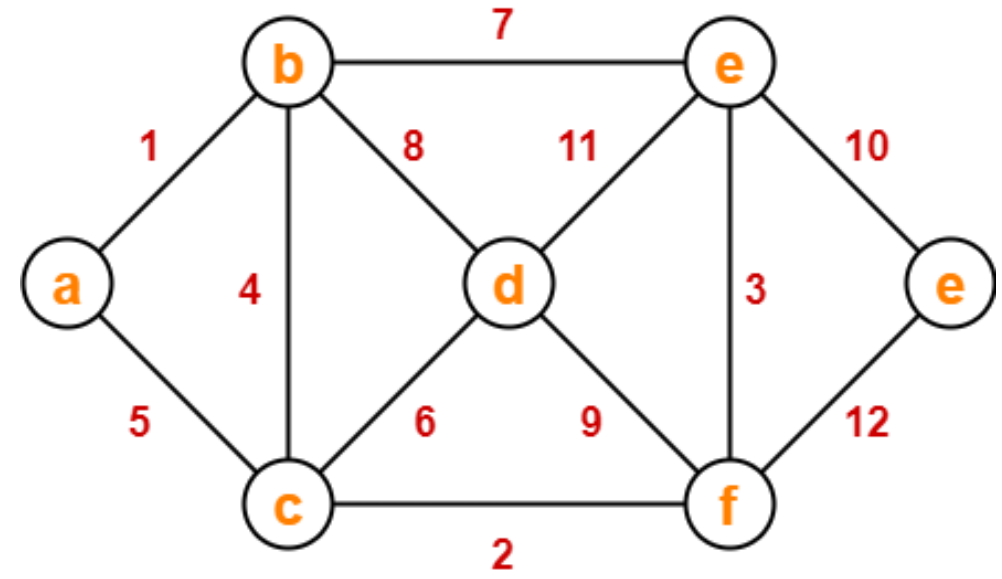
- Create a set **mstSet** that keeps track of vertices already included in MST.
- Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
- While **mstSet** doesn't include all vertices
    - Pick a vertex **u** that is not there in **mstSet** and has a minimum key value.
    - Include **u** in the **mstSet**.
    - Update the key value of all adjacent vertices of **u**. To update the key values, iterate through all adjacent vertices.
        - For every adjacent vertex **v**, if the weight of edge **u-v** is less than the previous key value of **v**, update the key value as the weight of **u-v**.

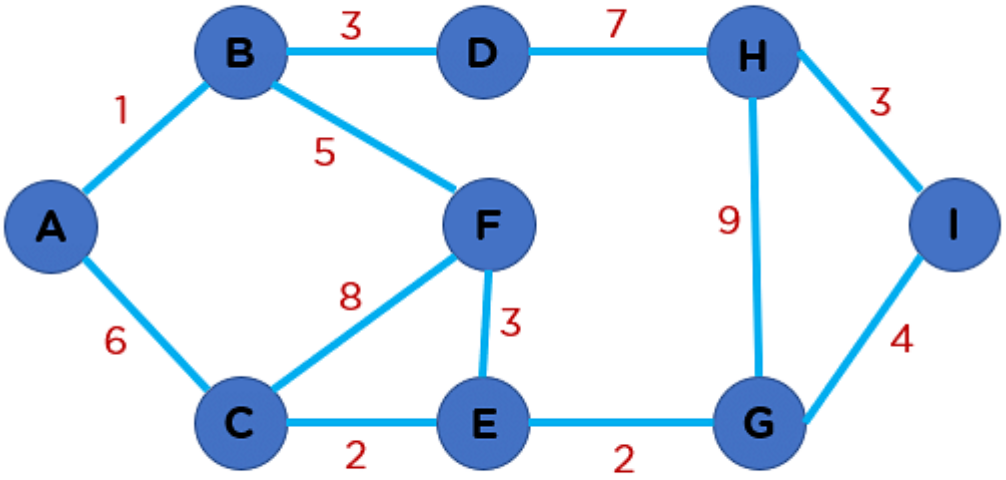
ALGORITHM Prim(G)

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = V,E
//Output: minimumspanning tree T of G
T ← {0} //the set of tree vertices can be initializedwith U U ← {1}
for i ← 1 to |V | − 1 do
let (u,v) be the lowest cost edge such that u€U and v € V-U
T ← T ∪ {(u,v)} // union add edge to spanning tree
U ← U ∪ {v}
Return T
End Prim

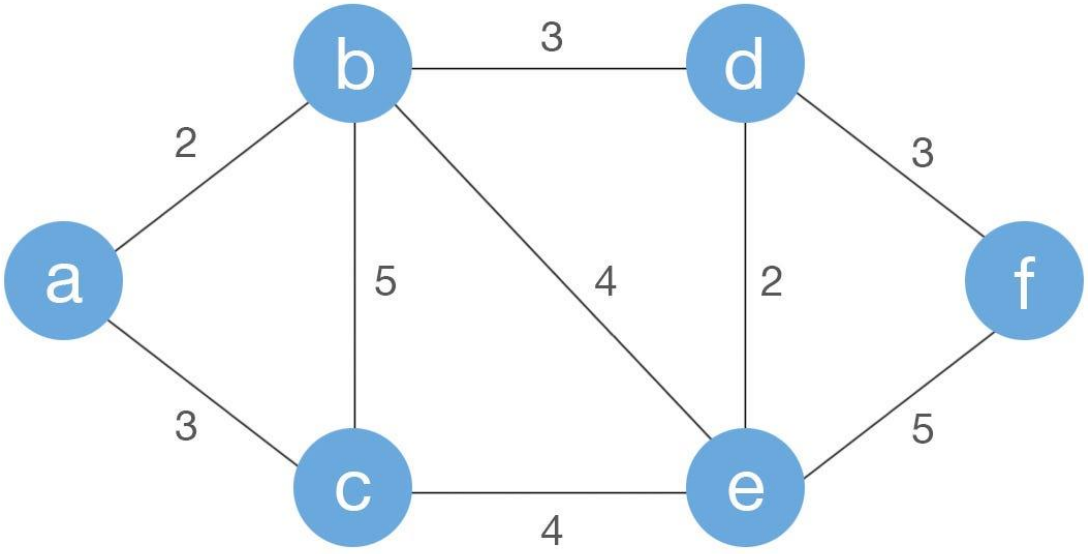
Other Examples for Practice:







Graph G(V, E)



2. Introduction to Kruskal's Algorithm:

Here we will discuss **Kruskal's algorithm** to find the MST of a given weighted graph.

In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and themaximum weighted edge at last. Thus we can say that it makes a locally **optimal choice** in each step in order to find the optimal solution. Hence this is a **Greedy Algorithm**.

How to find MST using Kruskal's algorithm?

Below are the steps for finding MST using Kruskal's algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

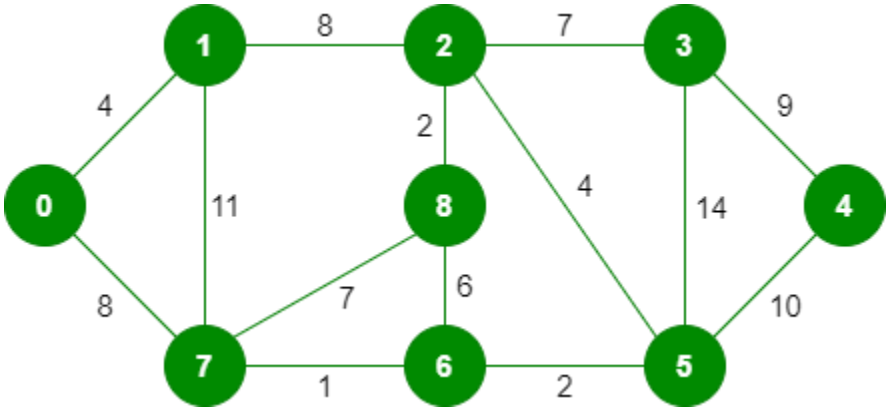
ALGORITHM Kruskal(G)

```
//Kruskal's algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph G = V, E

//Output: minimum spanning tree of G
T ← ∅;
 ecounter ← |V | − 1  //initialize the set of tree edges and its size

while T ≠ n-1 and ecounter ≠0 do
 k ← k + 1
     if  (u, v) does not form a cycle
                T ← T ∪ {u, v}     // add  (u, v) into T
     Delete (u, v) from min heap
    ecounter ← ecounter + 1
 return T
 end Kruskal
```

Illustration:

Below is the illustration of the above approach:

Input Graph:



The graph contains 9 vertices and 14 edges. So, the minimum spanning treeformed will be having (9 – 1) = 8 edges.

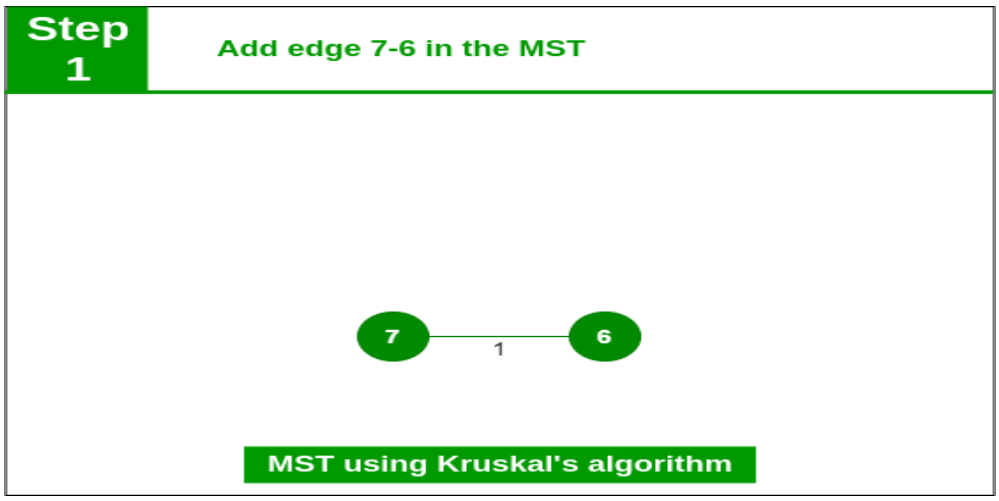
After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6

Weight	Source	Destination
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

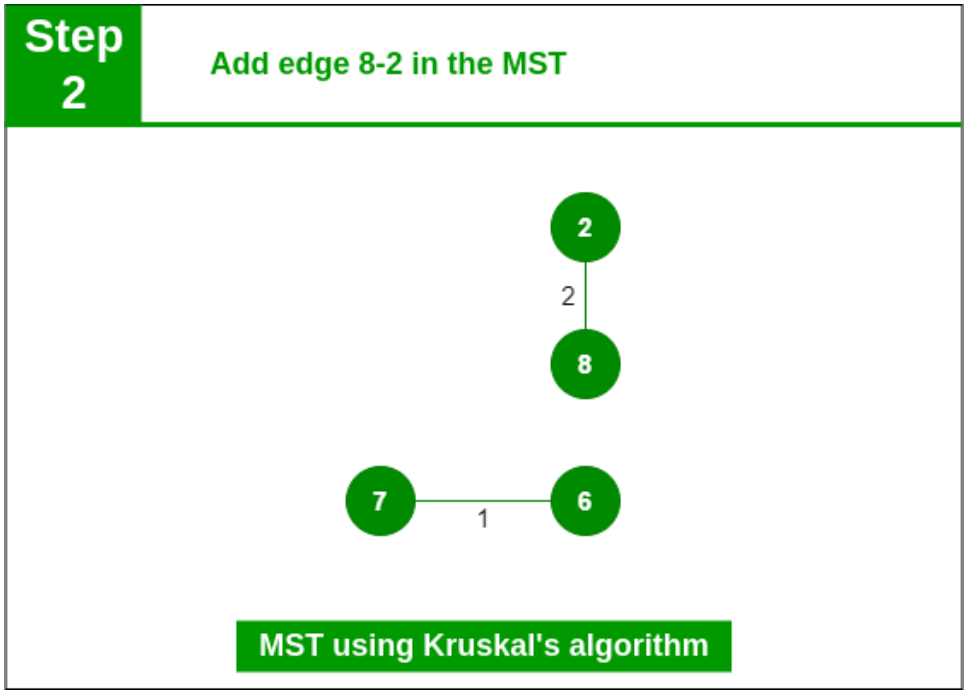
Now pick all edges one by one from the sorted list of edges

Step 1: *Pick edge 7-6. No cycle is formed, include it.*



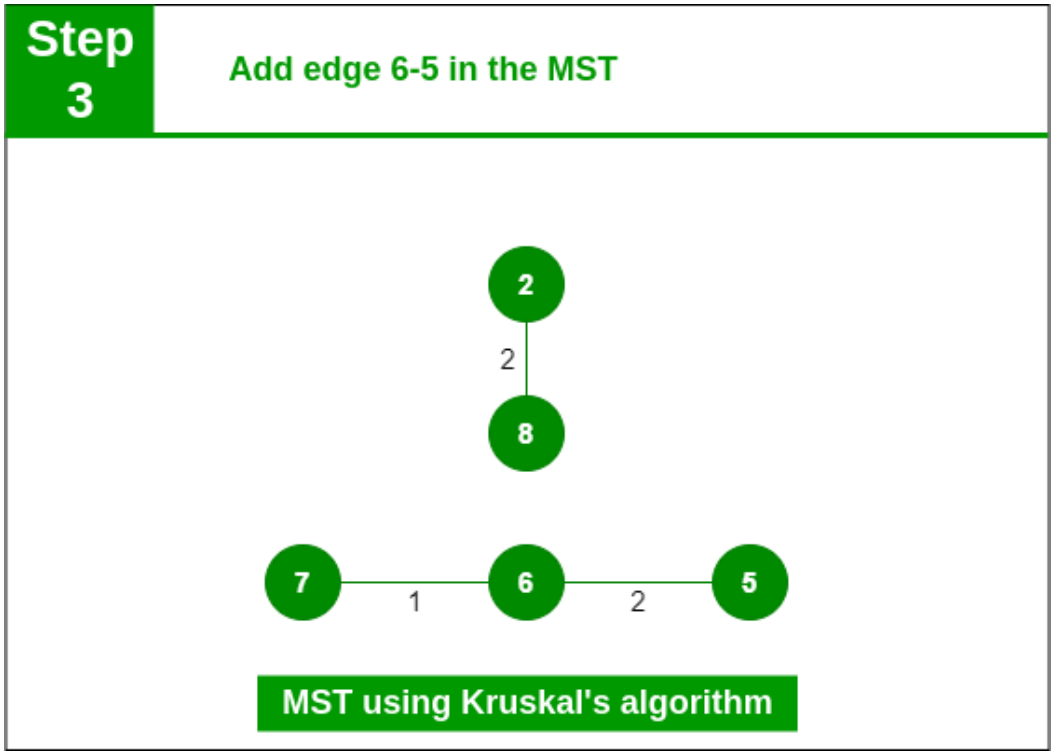
Add edge 7-6 in the MST

Step 2: *Pick edge 8-2. No cycle is formed, include it.*



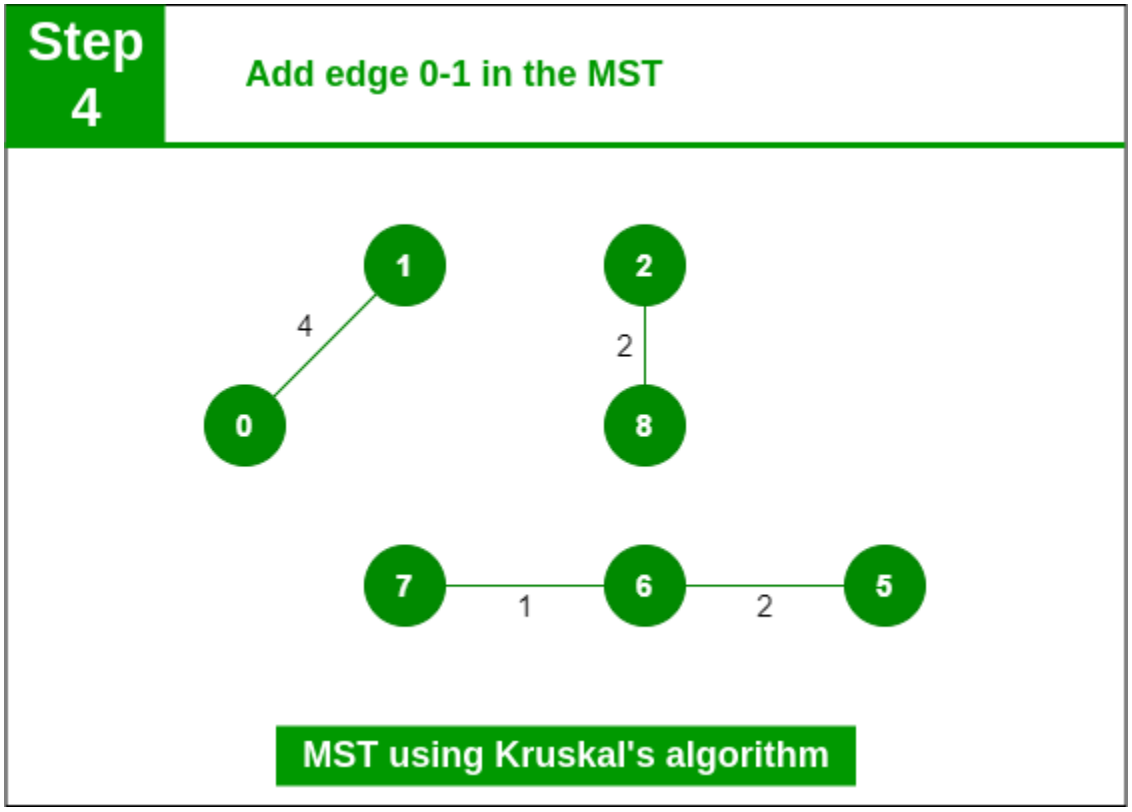
Add edge 8-2 in the MST

Step 3: *Pick edge 6-5. No cycle is formed, include it.*



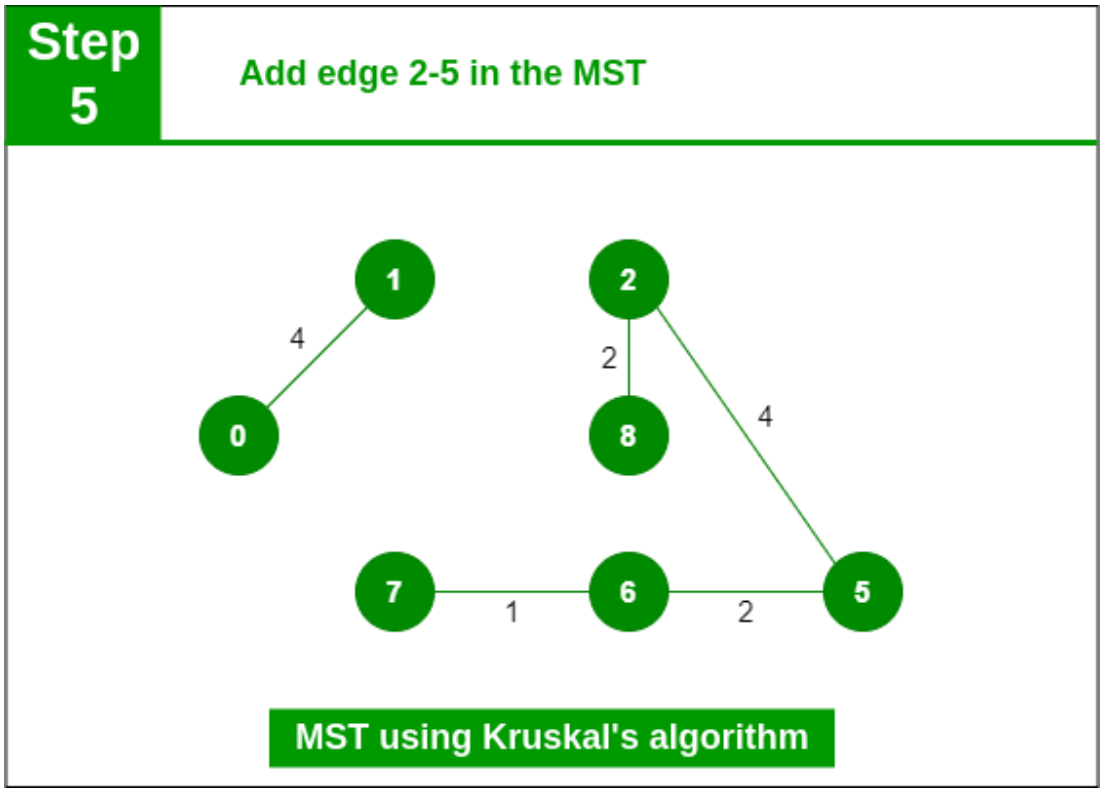
Add edge 6-5 in the MST

Step 4: *Pick edge 0-1. No cycle is formed, include it.*



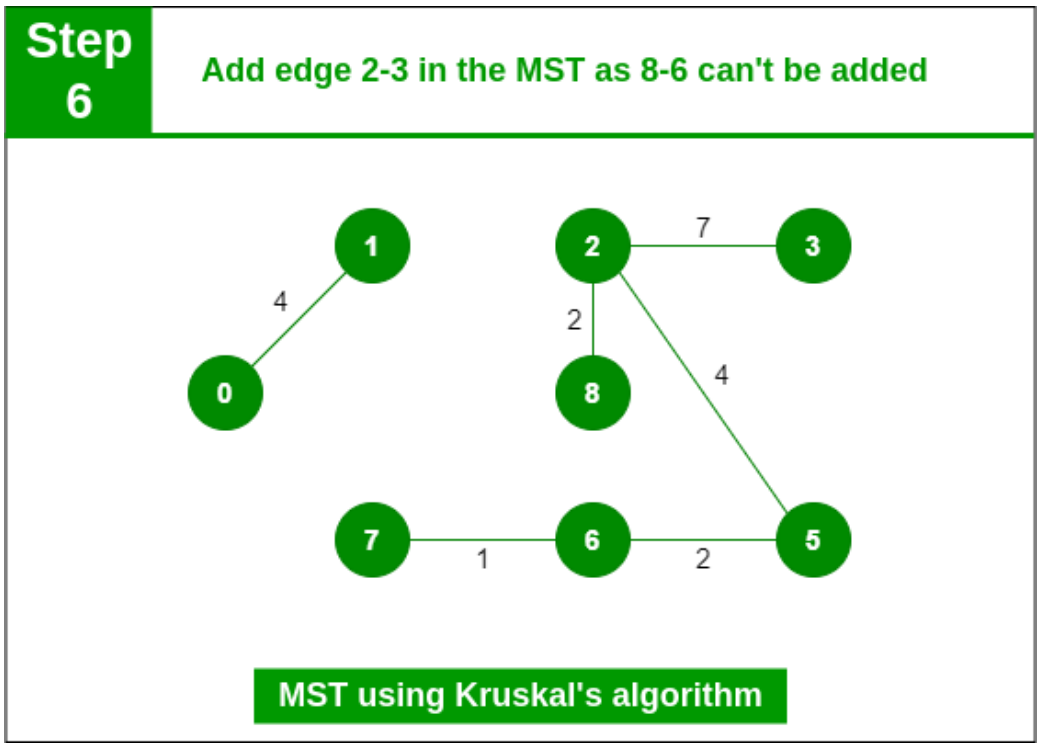
Add edge 0-1 in the MST

Step 5: *Pick edge 2-5. No cycle is formed, include it.*



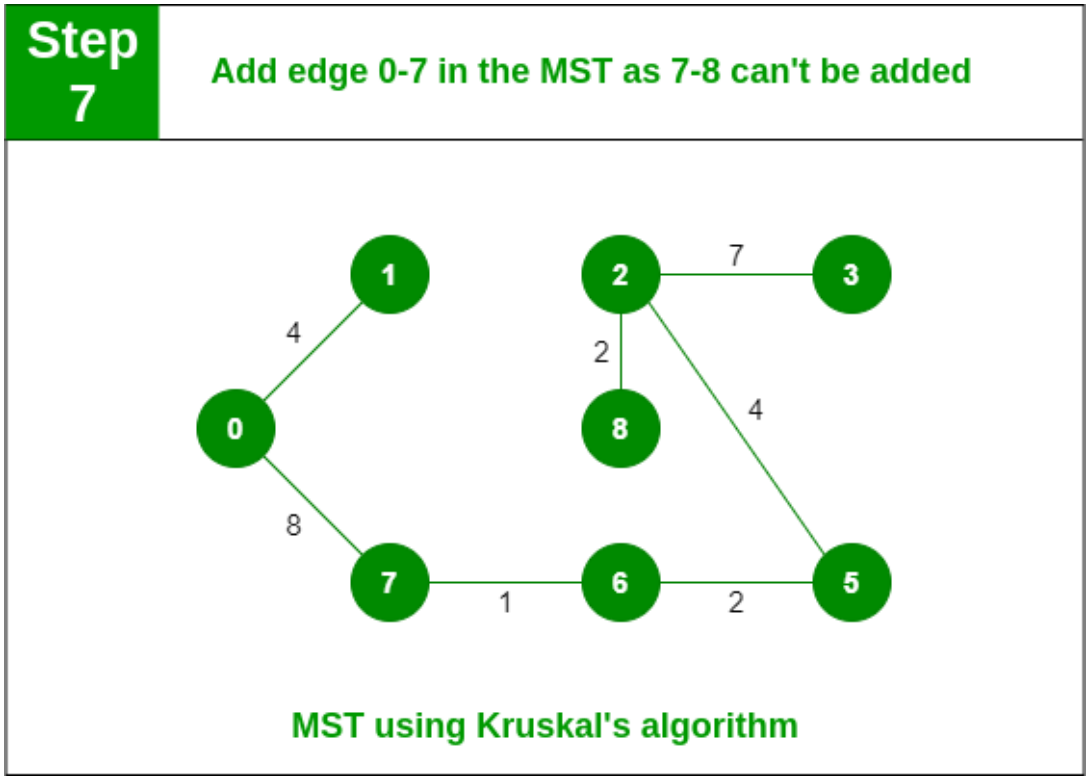
Add edge 2-5 in the MST

Step 6: *Pick edge 8-6. Since including this edge results in the cycle, discardit. Pick edge 2-3: No cycle is formed, include it.*



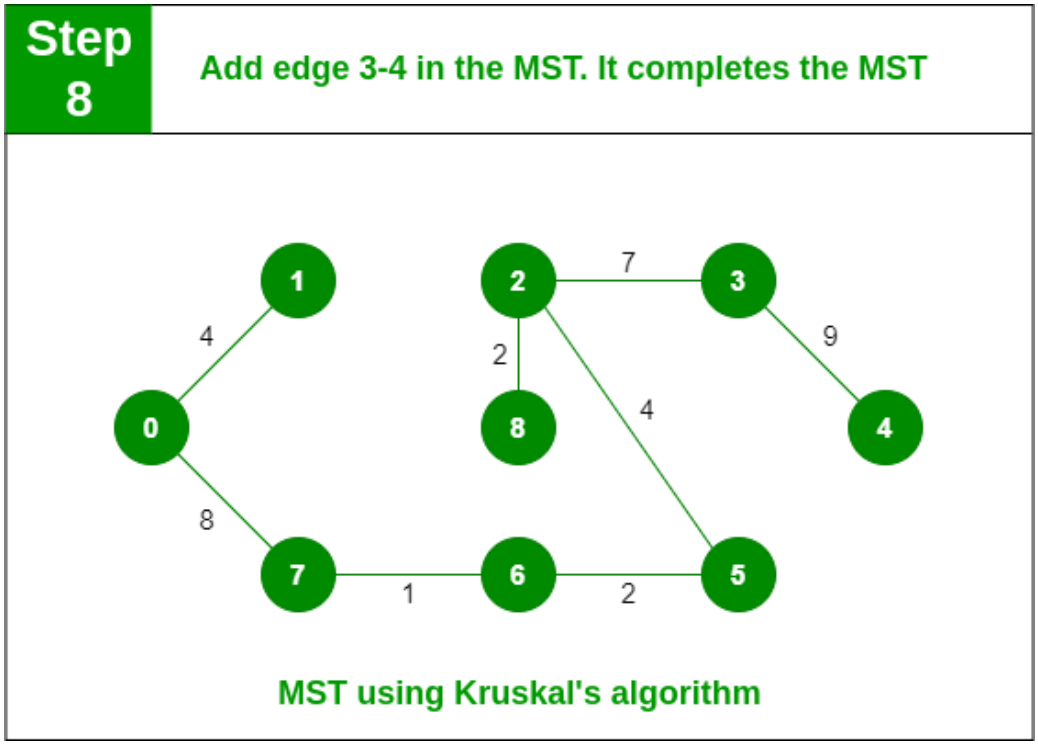
Add edge 2-3 in the MST

Step 7: *Pick edge 7-8. Since including this edge results in the cycle, discardit. Pick edge 0-7. No cycle is formed, include it.*



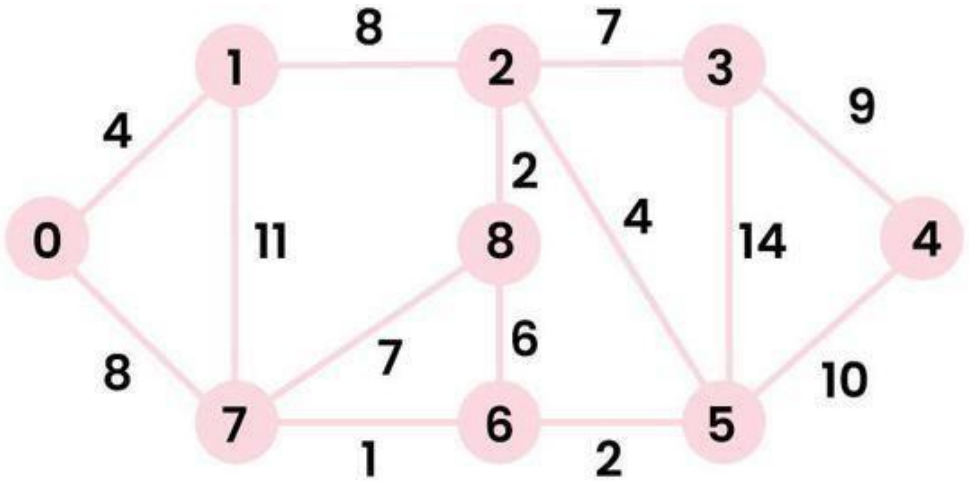
Add edge 0-7 in MST

Step 8: *Pick edge 1-2. Since including this edge results in the cycle, discardit. Pick edge 3-4. No cycle is formed, include it.*



Dijkstra'sAlgorithm,

Given a weighted graph and a source vertex in the graph, find the **shortest paths** fromthe source to all the other vertices in the given graph.
Note: The given graph does not contain any negative edge.
Examples:
Input: *src = 0, the graph is shown below.*



Working of Dijkstra's Algorithm

Output: *0 4 12 19 21 11 9 8 14*

Explanation:
The distance from 0 to 1 = 4.

The minimum distance from 0 to 2 = 12. 0->1->2

The minimum distance from 0 to 3 = 19. 0->1->2->3

The minimum distance from 0 to 4 = 21. 0->7->6->5->4

The minimum distance from 0 to 5 = 11. 0->7->6->5

The minimum distance from 0 to 6 = 9. 0->7->6

The minimum distance from 0 to 7 = 8. 0->7

The minimum distance from 0 to 8 = 14. 0->1->2->8

What is a Decision Tree?

A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote theresult of the algorithm. It is a versatile supervised machine-learning algorithm,which is used for both classification and regression problems. It is one of the very powerful algorithms. And it is also used in Random Forest to train on different subsets of training data, which makes random forest one of the mostpowerful algorithms in machine learning.

Decision Tree Terminologies

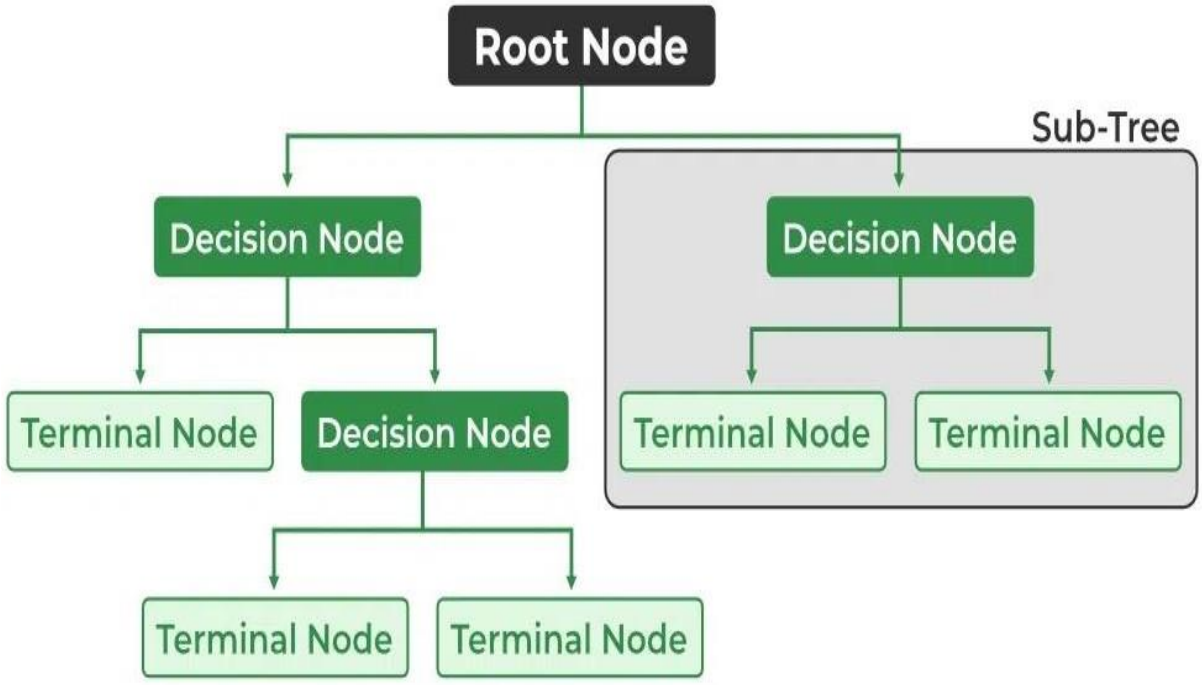
Some of the common Terminologies used in Decision Trees are as follows:

- **Root Node:** It is the topmost node in the tree, which represents the complete dataset. It is the starting point of the decision-making process.
- Decision/Internal Node: A node that symbolizes a choice regarding an input feature. Branching off of internal nodes connects them to leaf nodes or other internal nodes.
- **Leaf/Terminal Node:** A node without any child nodes that indicates a class label or a numerical value.
- **Splitting:** The process of splitting a node into two or more sub-nodes usinga split criterion and a selected feature.
- **Branch/Sub-Tree:** A subsection of the decision tree starts at an internal node and ends at the leaf nodes.

- **Parent Node:** The node that divides into one or more child nodes.
- **Child Node:** The nodes that emerge when a parent node is split.
- **Variance**: Variance measures how much the predicted and the target variables vary in different samples of a dataset. It is used for regression problems in decision trees. **Mean squared error, Mean Absolute Error, friedman_mse, or Half Poisson deviance** are used to measure the variance for the regression tasks in the decision tree.

- **Information Gain:** Information gain is a measure of the reduction in impurity achieved by splitting a dataset on a particular feature in a decision tree. The splitting criterion is determined by the feature that offers the greatest information gain, It is used to determine the most informative feature to split on at each node of the tree, with the goal of creating pure subsets

- **Pruning**: The process of removing branches from the tree that do not provide any additional information or lead to overfitting



P, NP, CoNP, NP hard and NP complete

In computer science, there exist some problems whose solutions are not yet found, the problems are divided into classes known as **Complexity Classes**. In complexity theory, a Complexity Class is a set of problems with related complexity. These classes help scientists to group problems based on how much time and space they require to solve problems and verify the solutions. It is the branch of the theory of computation that deals with the resources requiredto solve a problem.

The common resources are time and space, meaning how much time the algorithm takes to solve a problem and the corresponding memory usage.

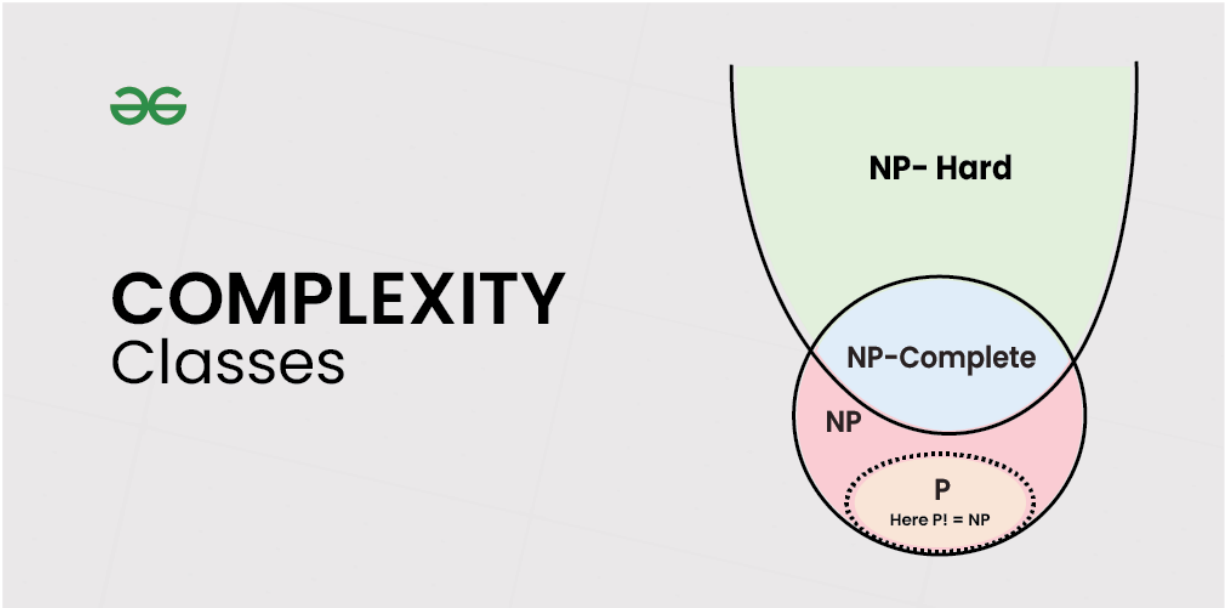
- The time complexity of an algorithm is used to describe the number of steps required to solve a problem, but it can also be used to describe how long it takes to verify the answer.
- The space complexity of an algorithm describes how much memory is required for the algorithm to operate.

Complexity classes are useful in organising similar types of problems.

Types of Complexity Classes

This article discusses the following complexity classes:

1. **P Class**
2. **NP Class**
3. **CoNP Class**
4. **NP-hard**
5. **NP-complete**



P Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a "yes" or "no" answer) that can be solved by a deterministic machine in polynomial time.

Features:

- The solution to **P problem**s is easy to find.
- **P** is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

This class contains many problems:

1. **Calculating the greatest common divisor.**
2. **Finding a maximum matching.**
3. **Merge Sort**

NP Class

The NP in NP class stands for **Non-deterministic Polynomial Time**. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Features:

- The solutions of the NP class are hard to find since they are being solved bya non-deterministic machine but the solutions are easy to verify.
- Problems of NP can be verified by a Turing machine in polynomial time.

Example:

Let us consider an example to better understand the **NP class**. Suppose thereis a company having a total of **1000** employees having unique employee **IDs**. Assume that there are **200** rooms available for them. A selection of **200** employees must be paired together, but the CEO of the company has the data of some employees who can't work in the same room due to personalreasons.

This is an example of an **NP** problem. Since it is easy to check if the given choice of **200** employees proposed by a coworker is satisfactory or not i.e. nopair taken from the coworker list appears on the list given by the CEO. But generating such a list from scratch seems to be so hard as to be completely impractical.

It indicates that if someone can provide us with the solution to the problem, we can find the correct and incorrect pair in polynomial time. Thus for the **NP** class problem, the answer is possible, which can be calculated in polynomial time.

This class contains many problems that one would like to be able to solve effectively:

1. **Boolean Satisfiability Problem (SAT).**
2. **Hamiltonian Path Problem.**
3. **Graph coloring.**

Co-NP Class

Co-NP stands for the complement of NP Class. It means if the answer to a problem in Co-NP is No, then there is proof that can be checked in polynomialtime.

Features:

- If a problem X is in NP, then its complement X' is also in CoNP.
- For an NP and CoNP problem, there is no need to verify all the answers at once in polynomial time, there is a need to verify only one particular answer"yes" or "no" in polynomial time for a problem to be in NP or CoNP.

Some example problems for CoNP are:

1. **To check prime number.**
2. **Integer Factorization.**

NP-hard class

An NP-hard problem is at least as hard as the hardest problem in NP and it is aclass of problems such that every problem in NP reduces to NP-hard.

Features:

- All NP-hard problems are not in NP.
- It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.
- A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Some of the examples of problems in Np-hard are:

1. **Halting problem.**
2. **Qualified Boolean formulas.**
3. **No Hamiltonian cycle.**

NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problemsare the hard problems in NP.

Features:

- NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.
- If one could solve an NP-complete problem in polynomial time, then one

could also solve any NP problem in polynomial time